DBMS 성능 평가를 위한 자동화된 벤치마크 관리기

심재희, 차상균

서울대학교 전기. 컴퓨터공학부
{jaehee, chask}@kdb.snu.ac.kr

Automated Benchmark Management Tool for DBMS Performance Evaluation

Jaehee Shim, Sang K. Cha
School of Electrical Engineering and Computer Science, Seoul National University

요 약

정보 통신 산업의 발달로 인해 데이터의 양이 점차 증가하고 고성능의 데이터 접근이 필요한 분야가 증가됨에 따라 DBMS의 성능에 대한 관심이 높아지고 있다. 따라서 DBMS 벤더들은 계속적으로 새로운 기술을 도입하여 확장 가능한 고성능의 시스템을 지원하는데 노력하고 있다. 그러나 새로운 기술에 대한 충분하고도 광범위한 테스트가 이뤄지지 않는다면 시스템의 성능과 안정성 등에 예상치 못한 문제가 생기기 마련이다. 따라서 벤치마크를 통하여 시스템의 취약점을 알아내고 기술의 질적 평가를 하는 과정이 필수적이지만, 벤치마크 프로그램을 작성하고 수행하는 과정에 상당히 오랜 시간이 걸리기 때문에 시스템에 대한 충분한 테스트를 빠르고 용이하게 하기가 힘들다. 이에 본 논문에서는 이런 문제점을 해결할 수 있는 방안으로 사용자의 특정한 응용 도메인에서 수행될 작업부하 프로그램을 간단히 작성할 수 있으며 그 외의 나머지 벤치마크 과정을 자동화하는, DBMS 성능 평가를 위한 자동화된 벤치마크 관리기를 설계하고 구현하였다. 본 논문에서 제안한 자동화된 DBMS 벤치마크 관리기를 사용하면 사용자는 간단한 코드 작성만으로도 응용 도메인의 벤치마크를 용이하게 할 수 있다.

1. 서론

정보 통신 산업의 발달로 인해 관리해야 할 데이터의 양이 점차 증가하고 고성능의 데이터 접근이 필요한 분야가 증가됨에 따라 DBMS의 성능에 대한 관심이 높아지고 있다. 특히 최근에는 갱신 성능이 뛰어난 P*TIME[1]과 같은 메모리 중심의 DBMS에 대한 신기술이 나오면서 DBMS 벤더들은 신기술을 도입한 고성능의 시스템을 지원하는데 주력하고 있다.

만약 기존의 아키텍처에 새로운 기술을 접목시킬 때 응용 기반에 대한 충분하고도 광범위한 테스트가 이루어지지 않는다면 시스템의 성능과 안정성 등에 예상치 못한 문제가 생기기 마련이다[2]. 따라서 신기술이 실제 어플리케이션에 어떤 영향을 끼치는지 테스트하는 과정과, 벤치마크를 통하여 시스템의 취약점을 알아내고 기술의 질적 평가를 하는 과정을 거쳐 시스템이 가지는 성능과 신뢰성, 확장성, 그리고 안정성 등을 사용자에게 입증시켜야 한다.

하지만 DBMS의 다양한 사용 용도에 따라 사용자마다 요구하는 응용 도메인이 모두 다르기 때문에 각 도메인의 특성에 맞춘 벤치마크 프로그램을 작성해야 한다[3]. 성능 평가를 위한 벤치마크에는 데이터베이스를 생성하는 프로그램과 여러 머신에서 원하는 만큼의 작업부하를 생성하고 이들을 중앙에서 제어하는 프로그램이 필요하다. 그리고 벤치마크를 수행할 응용 도메인의 특성에 맞춘 작업부하 프로그램이 필요하다. 그러나 사용자가 정의한 벤치마크 작업부하의 프로그램을 제외한 나머지 프로그램들은 성능 평가를 위한 벤치마크를 수행할 때마다 항상 같은 기능을 하게 되므로, 이를 자동화시키는 것이 효율적이다.

이에 본 논문에서는 위와 같은 문제점을 해결할 수 있는 방안으로 DBMS 성능 평가를 위한 자동화된 벤치마크 관리기(이하, DBMS 벤치마크 관리기)의 설계 및 구현을 하였다.

본 논문에서 논의될 DBMS 벤치마크 관리기는 사용자가 요구하는 응용 도메인의 특성에 맞춘 벤치마크를 수행할 때, 사용자는 단지 작업부하의 내용을 담은 프로그램의 작성만 수행하고, 나머지는 자동적으로 DBMS 벤치마크 관리기에서 처리하도록 설계하였다. 사용자가 기술하는 작업부하 프로그램 역시 측정할 벤치마크 레벨에 따라 클래스들간의 상속을 통한 추상 메소드들을 제공하여서 손쉽게 작성할 수 있도록 하였다.

DBMS 벤치마크 관리기는 이와 같이 성능 측정을 용이하게 할 수 있는 환경을 사용자에게 제공하여 DBMS의 벤치마크를 빠르고 편리하게 수행할 수 있도록 하는 것에 초점을 맞추었다.

본 논문의 구성은 다음과 같다. 2절에서는 DBMS 벤치마크 관리기의 전체적인 구조를 벤치마크의 수행 순서에 따라 설명하며, 3절에서는 사용자가 작업부하 프로그램을 어떤 방법으로 작성하도록 설계했는지 설명하고 예제를 보인다. 마지막으로 4절에서 결론을 맺는다.

2. DBMS 벤치마크 관리기의 구조

본 절에서는 DBMS 벤치마크 관리기의 전체적인 구조를 벤치마크의 수행 순서에 따라 설명하도록 한다.

일반적으로 DBMS의 벤치마크를 수행하려면, 먼저 주어진 스키마에 따라 데이터베이스를 생성하고 벤치마크 테스트

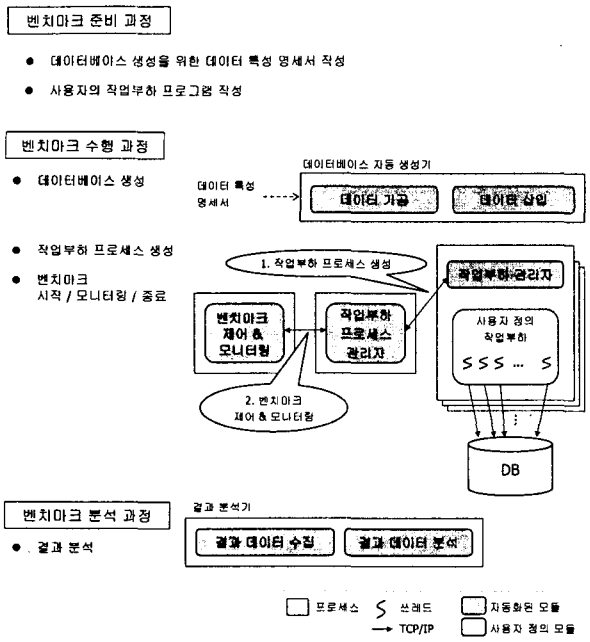

그림 1. DBMS 벤치마크 관리기를 이용하여 나타낸 벤치마크의 수행 단계

시나리오에 따라 작업 부하를 정의하는 준비 과정을 거친다. 그 후, DBMS에 충분한 부하를 걸어주면서 DBMS의 성능과 시스템 자원 활용을 모니터 하게 되며, 마지막으로 시스템 자원의 활용도와 DBMS의 성능 측정치를 비교하여 분석하는 과정을 거치게 된다. 이와 같은 일반적인 벤치마크 수행 과정을 용이하게 하는 DBMS 벤치마크 관리기의 구조는 그림 1과 같다.

벤치마크의 준비 단계에서 사용자는 데이터베이스 생성을 위한 데이터 특성의 명세서를 작성하고, 작업부하 프로그램을 작성하게 된다.

사용자는 데이터베이스의 생성을 위해 그림 2와 같이 데이터 타입, 데이터 값의 범위, 균일 분포 혹은 가우시안 분포와 같은 통계적인 특성을 명시한 파일을 작성하게 된다. 다음으로는 사용자가 작업부하 프로그램을 작성하게 된다. DBMS 벤치마크 관리기에서는 작업부하 프로그램을 손쉽게 작성할 수 있도록 클래스의 상속을 통한 추상 메소드를 제공하는데 이에 관한 자세한 사항은 3절에서 자세하게 기술한다.

벤치마크의 수행 단계는 데이터베이스의 생성, 작업부하 프로세스의 생성, 벤치마크의 시작, 모니터링, 그리고 종료하는 과정으로 이루어 진다.

데이터베이스의 생성에서는, 사용자가 작성한 데이터의 특성 명세서를 바탕으로 데이터를 생성하고 이를 DB에 삽입하는 전 과정이 데이터베이스 자동 생성기를 통하여 이루어 진다. 이를 위하여 통계적인 특성에 따라 인공적으로 데이터를 가공하는 모듈과 DBMS에 데이터를 삽입하는 모듈을 지원한다. 통계적인 특성으로는 균일 분포, 가우시안 분포, 그리고 Zipf 분포 방법[4]을 지원한다.

DBMS의 일반적인 성능 평가는 다수의 클라이언트를 생성하여 DBMS에 작업을 요청하고 그 응답 성능을 측정하는 것이므로, 충분한 부하를 걸어주기 위하여 작업부하 프로세스를 여러 개의 머신에서 동시에 실행시키는 것이 효과적이다.

```
Name=id
Type=int
RangeBegin=0
RangeEnd=100
Distribution=gaussian
...
```

그림 2. 데이터 특성 명세서

이를 위하여 사용자가 벤치마크 제어기에서 브로드캐스팅을 통하여 각기 다른 머신의 작업부하 프로세스들을 관리할 수 있도록 하였다. 이는 작업부하 프로세스 관리자 모듈에서 지원한다. 작업부하 프로세스 관리자는 하나 이상의 작업부하 프로세스를 생성하는데, 왜냐하면 프로세스 당 생성할 수 있는 스레드 수에 한계가 있기 때문이다.

벤치마크가 시작되면 사용자는 DBMS가 초당 처리하는 트랜잭션 수(TPS)와 응답 시간 등의 성능과 시스템 자원 사용량을 DBMS 벤치마크 관리기 내의 모니터링 툴을 사용하여 모니터 하게 된다. 성능은 작업부하 프로세스 관리자에서 각 작업부하 프로세스들의 수행 결과를 합한 후, 벤치마크 모니터에서 각 머신마다 측정된 수행 횟수를 최종적으로 합하여 산출된다. 모니터링 후 벤치마크는 종료된다.

벤치마크가 종료되면 결과를 분석하는 과정을 수행하게 된다. 이는 결과 분석기라는 프로세스에서 수행하게 되는데, 결과 데이터를 수집하고 분석을 수행하는 두 모듈로 구성되어 있다. 결과 데이터 수집하는 모듈에서는 벤치마크를 모니터하는 동안 측정된 성능 값과 시스템의 자원 사용량을 시스템 시간과 함께 파일에 기록하도록 하였다. 분석 수행 모듈에서는 이렇게 수집된 벤치마크 결과값으로부터 평균, 표준 분포 등을 나타낼 수 있도록 하였다.

3. 사용자의 작업부하 프로그램 작성

DBMS 벤치마크 관리기는 사용자가 작업부하 프로그램을 용이하게 작성할 수 있도록 DBMS의 벤치마크 레벨을 정의하여 작업부하 클래스 계층을 형성하였다. 본 절에서는 이를 예제를 통하여 설명하도록 한다.

3.1. 작업부하 클래스 계층

DBMS가 SQL과 JDBC를 지원하므로 DBMS 벤치마크의 수행 레벨을 다음과 같이 분류할 수 있다.

I. SQL 질의 레벨
하나 이상의 SQL 형태의 질의로 구성되며, 각 질의마다 주어진 수행 비율로 DBMS에 질의 처리 요청을 하는 벤치마크 시나리오를 구성한다. 최하위 레벨이다.

II. 트랜잭션 레벨
하나 이상의 SQL 형태의 질의로 구성되며, 각 질의는 사용자가 기술한 제어 흐름이나 앞서 수행한 질의 결과값 등에 따라 수행 순서나 연관 관계가 정의되어 하나의 트랜잭션을 구성한다.

III. 어플리케이션 레벨
최상위 레벨로써, 하나 이상의 트랜잭션으로 구성된다. 사용자가 정의한 트랜잭션들 간의 제어 흐름에 따라 하나의 어플리케이션을 이루게 된다.

위와 같이 분류한 세 가지 벤치마크 레벨에 기초하여 DBMS 벤치마크 관리기는 그림 3과 같은 작업부하 클래스 계층 관계를 가진다. 작업 클래스는 사용자가 지정한 작업부하 클래스를 생성하고 수행한다.

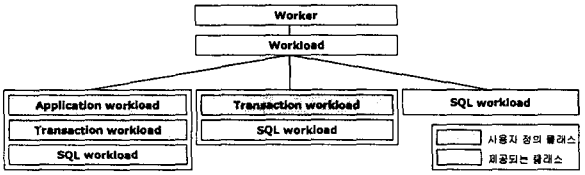

그림 3. 작업부하 클래스들의 계층 관계

SQL 질의 레벨은 질의문과 변수값에 대한 정보만 주어지면 질의를 처리할 수 있는 간단한 구조를 가지고 있다. 따라서 SQL 질의 작업부하 클래스에서는 질의를 수행하는데 필요한 prepared statement, 변수값 세팅, 질의 수행, 결과값 체크를 하는 메소드들을 제공하여, 사용자가 프로그램을 작성할 필요 없이 단지 질의문과 관련 변수 정보들을 입력하면 질의를 처리할 수 있도록 하였다.

트랜잭션은 하나 이상의 질의로 이루어져 있으므로, 트랜잭션 작업부하 클래스는 SQL 질의 작업부하 클래스의 오브젝트를 질의 개수만큼 생성하며, 이 클래스에서 제공하는 메소드를 사용하여 하나의 트랜잭션 처리 루틴을 사용자가 쉽게 작성할 수 있다.

어플리케이션 작업부하 클래스는 여러 개의 트랜잭션 작업부하 클래스 오브젝트를 생성하며, 각각의 트랜잭션 작업부하 클래스 오브젝트는 SQL 질의 작업부하 클래스 오브젝트들을 생성하게 되는 구조를 가지고 있다. 그리고 각 트랜잭션 처리 루틴은 위와 마찬가지로 SQL 질의 작업부하 클래스에서 제공하는 메소드를 사용하여 구성하게 된다. 어플리케이션 작업부하 클래스는 트랜잭션들 끼리의 관계로 구성되기 때문에, 한 트랜잭션의 결과값으로부터 그 다음 트랜잭션의 입력값이 결정되는 등의 경우가 발생할 수 있다. 트랜잭션의 입출력 값의 형태는 각 트랜잭션마다 다르게 정의될 수 있기 때문에 사용자가 해당 트랜잭션의 입출력 형태를 정의한 클래스를 작성해야 한다.

작업 클래스에서는 이러한 작업부하 클래스의 오브젝트들을 사용자가 원하는 개수만큼 생성시킬 수 있으며, 생성된 각 작업부하 오브젝트들은 주어진 수행 비율과 순서로 수행되도록 하였다.

3.2. 작업부하 클래스의 코드

본 절에서는 범용 벤치마크 테스트인 TPC-C의 주요 트랜잭션 중의 하나인 new order 트랜잭션을 수행하는 작업부하 클래스를 예로 들어 작업부하 클래스의 구조를 설명하도록 한다. 이는 그림 4와 같은 pseudo 코드를 가질 수 있다.

상단의 코드는 작업 클래스의 코드이다. **start** 함수가 호출되면 해당하는 벤치마크 레벨의 작업부하 클래스 오브젝트가 생성되며 각 오브젝트마다 스레드도 생성된다. 하단 왼쪽의 코드는 작업부하 클래스이며 추상 메소드를 선언하였다. 그 메소드로는 DB 연결 및 prepared statement 등의 초기화를 수행하는 **init**, 변수 값을 생성하고 질의문을 실행시키는 **execute**, 그리고 DB 연결을 끊는 작업 등을 수행하는 **finalize** 메소드를 들 수 있다. 하단 오른쪽은 사용자가 직접 작성하는 new order 트랜잭션 작업부하 클래스의 코드이다. 내부에 SQL 질의 작업부하 클래스 오브젝트들을 생성하고 그 오브젝트들이 제공하는 메소드를 사용하여 수행할 트랜잭션을 **execute** 메소드 안에서 정의한 것을 확인할 수 있다.

4. 결론

```
< DBMS 벤치마크 관리자에서 제공하는 클래스 >
class Worker
{
    Workload[] workload;
    String classFileName;
    void start() {
        Class class = ClassLoader.loadClass(classFileName); // 사용자가 작성한 클래스 로딩
        switch (벤치마크 레벨)// 벤치마크 레벨에 따른 해당 클래스 오브젝트 생성
            case 'SQL 질의 레벨':
                for (i=0; i<nWorkloads; i++) { // SQL workload 스레드 생성
                        workload[i] = new SQLWorkload();
                        for (j=0; j<nThreads; j++)
                                workload[i].start();    // 스레드 시작
                }
            case '트랜잭션 레벨 or 어플리케이션 레벨':
                for (i=0; i<nWorkloads; i++) { // 트랜잭션, 어플리케이션 workload 스레드 생성
                        workload[i] = class.newInstance(); // 스레드 생성
                        for (j=0; j<nThreads; j++)
                                workload[i].start();    // 스레드 시작
                }
    }
    void stop();      // 스레드 비활성화
    void monitor();   // 각 스레드의 모니터링

    void run() {      // 스레드 body
        workload.init();
        while (thread is active)
                workload.execute();
        workload.finalize();
    }
}
```

```
<DBMS 벤치마크 관리자에서 제공하는 클래스>
Abstract Class Workload
{
    abstract void init();
    abstract void execute();
    abstract void finalize();
}
```

```
<사용자가 작성한 new order 트랜잭션 클래스>
Class NewOrderTransaction
extends Workload
{   // SQL 질의 작업부하 클래스 오브젝트 생성
    QueryWorkload[] newOrderQuery;

    void init() { // prepared statement
            for (i=0; i<nQuery; i++)
                newOrderQuery[i].prepareQuery();
    }
    void execute() {
            // 주문 header를 생성한다.
            newOrderQuery[0].setQuery();
            newOrderQuery[0].executeQuery();
            ...
            // 품목들을 주문한다.
            for (# of items in the order) {
            newOrderQuery[5].setQuery();
            newOrderQuery[5].executeQuery();
            ...
            }
    }
    void finalize();
}
```

그림 4. 트랜잭션 레벨 작업 부하 클래스의 코드

본 논문에서는 사용자가 요구하는 응용 도메인의 특성에 맞춘 벤치마크를 용이하게 수행할 수 있는 자동화된 DBMS 벤치마크 관리기의 설계 및 구현에 대하여 기술하였다.

사용자에게 추상 메소드를 제공하여 작업부하 프로그램을 쉽게 작성하도록 하며, 나머지 벤치마크 과정은 자동화하여 DBMS의 복잡한 벤치마크 수행을 빠르고 간단하게 하는데 초점을 맞추었다.

참고 문헌

[1] 차상균, 송창빈, 유승원 및 Transact In Memory 프로젝트 팀, "P*TIME: 고성능 메인 메모리 DBMS 구조와 활용", *Korean DataBase Conference* 2001

[2] M.L. Kersten, F. Kwakkel, "Design and Implementation of a DBMS Performance Assessment Tool", In *Proceedings of the 4th international DEXA Conference* 1993

[3] D. Chays, S. Dan, P.G.. Frankl, F.I. Vokolos, and E.J. Weyuker, "A Framework for Testing Database Applications", In *Proceedings of ISSTA* 2000

[4] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P.J. Weinberger, "Quickly generating billion-record synthetic databases", *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):243-252, June 1994.

[5] The Grinder, http://grinder.sourceforge.net

[6] Wily Technology, http://www.wilytech.com